Ordenação

Sérgio Canuto - sergio.canuto@ifg.edu.br

Definições - Ordenação e Busca

- A ordenação nada mais é do que o ato de colocar um conjunto de dados em determinada ordem predefinida, como mostra o exemplo a seguir:
  - 5, 2, 1, 3, 4: FORA DE ORDEM.
  - 1, 2, 3, 4, 5: ORDENADO.
- A ordenação permite que o acesso aos dados seja feita de forma mais eficiente.
- A ordenação de um conjunto de dados é feita utilizando como base uma chave específica.
  - A chave de ordenação é o “campo” do item utilizado para comparação. É por meio dele que sabemos se determinado elemento está à frente ou não de outros no conjunto ordenado.
  - Chaves:

    Numérica: 1, 2, 3, 4, 5.

    Lexicográfica (ordem alfabética): Ana, André, Bianca, Ricardo.

Definições - Ordenação e Busca

Ordenação ser classificados como de **ordenação interna** (**in-place**) ou **externa**:

- **Ordenação interna**: o conjunto de dados a ser ordenado cabe todo na memória principal. Qualquer elemento pode ser imediatamente acessado.
- **Ordenação externa**: o conjunto de dados a ser ordenado não cabe na memória principal (está armazenado em memória secundária, por exemplo, em um arquivo). Os elementos são acessados sequencialmente ou em grandes blocos.

Definições - Ordenação e Busca

- Um algoritmo de ordenação é considerado estável se a ordem dos elementos com chaves iguais não muda durante a ordenação.
- Exemplo:
  - 5a, 2, 5b, 3, 4, 1: **dados não ordenados.**
- Um algoritmo de ordenação será considerado **estável** se o valor **5a** vier antes do valor **5b** quando esse conjunto de dados for ordenado de forma crescente, ou seja, o algoritmo preserva a ordem relativa original dos valores:
  - 1, 2, 3, 4, **5a**, **5b**: **ordenação estável.**
  - 1, 2, 3, 4, **5b**, **5a**: **ordenação não estável.**

Questão :

Critique o código da seguinte função, que promete decidir se o vetor `v[0..n-1]` está em ordem crescente.

```
int verifica (int v[], int n) {
   if (n > 1)
      for (int i = 1; i < n; i++)
         if (v[i-1] > v[i]) return 0;
      return 1; }
```

ALGORITMOS BÁSICOS DE ORDENAÇÃO - Ordenação por “bolha”: bubble sort

- Algoritmo bem conhecido
- Idéia de bolhas flutuando em um tanque de água em direção ao topo, até encontrarem o seu próprio nível
- Movimenta, uma posição por vez, o maior valor existente na porção não ordenada de um array para a sua respectiva posição no array ordenado.
- Isso é repetido até que todos os elementos estejam nas suas posições correspondentes.
- O princípio de funcionamento deste algoritmo é a troca de valores em posições consecutivas de array para que, desse modo, fiquem na ordem desejada.

ALGORITMOS BÁSICOS DE ORDENAÇÃO - Ordenação por “bolha”:

bubble sort -- Versão 1.0

```
void BubbleSort(int vetor[], int n) {
  int i,j,aux;

  for(i=N-1;i>0;i--)
    for(j=0;j<i;j++)
      if(vetor[j] > vetor[j+1]) {
        aux = vetor[j];
        vetor[j] = vetor[j+1];
        vetor[j+1] = aux;
      }
}
```

ALGORITMOS BÁSICOS DE ORDENAÇÃO - Ordenação por “bolha”:

bubble sort -- Versão 1.0

```
void BubbleSort(int vetor[], int n) {
  int i,j,aux;

  for(i=N-1;i>0;i--)
    for(j=0;j<i;j++)
      if(vetor[j] > vetor[j+1]) {
        aux = vetor[j];
        vetor[j] = vetor[j+1];
        vetor[j+1] = aux;
      }
}
```

Inclua uma variável “flag” para melhorar o algoritmo bolha no melhor caso!

```
void BubbleSort2(int vetor[], int n) {
  int i,j,aux,troca;
  for(i=N-1;i>0;i--) {
    troca = 0;
    for(j=0;j<i;j++)
      if(vetor[j] > vetor[j+1]) {
        aux = vetor[j];
        vetor[j] = vetor[j+1];
        vetor[j+1] = aux;
        troca = 1;
      }
    if (troca==0) break;
}
```

Se nenhuma troca é realizada em um determinado passo, então todos os elementos já se encontram ordenados.

ALGORITMOS BÁSICOS DE ORDENAÇÃO - Ordenação por “bolha”: bubble sort

Método bubble sort

```
01  void bubbleSort(int *V  , int N){
02      int i, continua, aux, fim = N;
03      do{
04          continua = 0;
05          for(i = 0; i < fim-1; i++){
06              if(V[i] > V[i+1]){//anterior > próximo? Mudar!
07                  aux = V[i];
08                  V[i] = V[i+1];
09                  V[i+1] = aux;
10                  continua = i;
11              }
12          }
13          fim--;
14      }while(continua != 0);
15  }
```

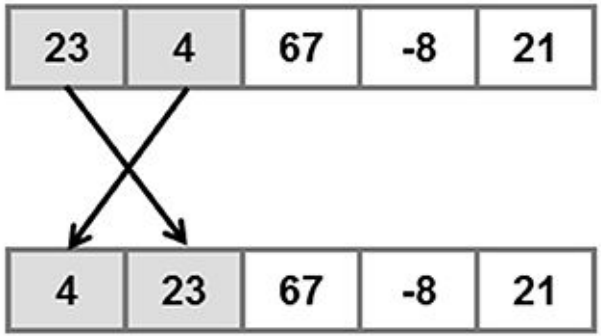
ALGORITMOS BÁSICOS DE ORDENAÇÃO - Ordenação por “bolha”: bubble sort

Animação:

https://pt.wikipedia.org/wiki/Ficheiro:Bubble_sort_animation.gif

https://www.youtube.com/watch?v=lyZQPjUT5B4

ALGORITMOS BÁSICOS DE ORDENAÇÃO - Ordenação por “bolha”: bubble sort



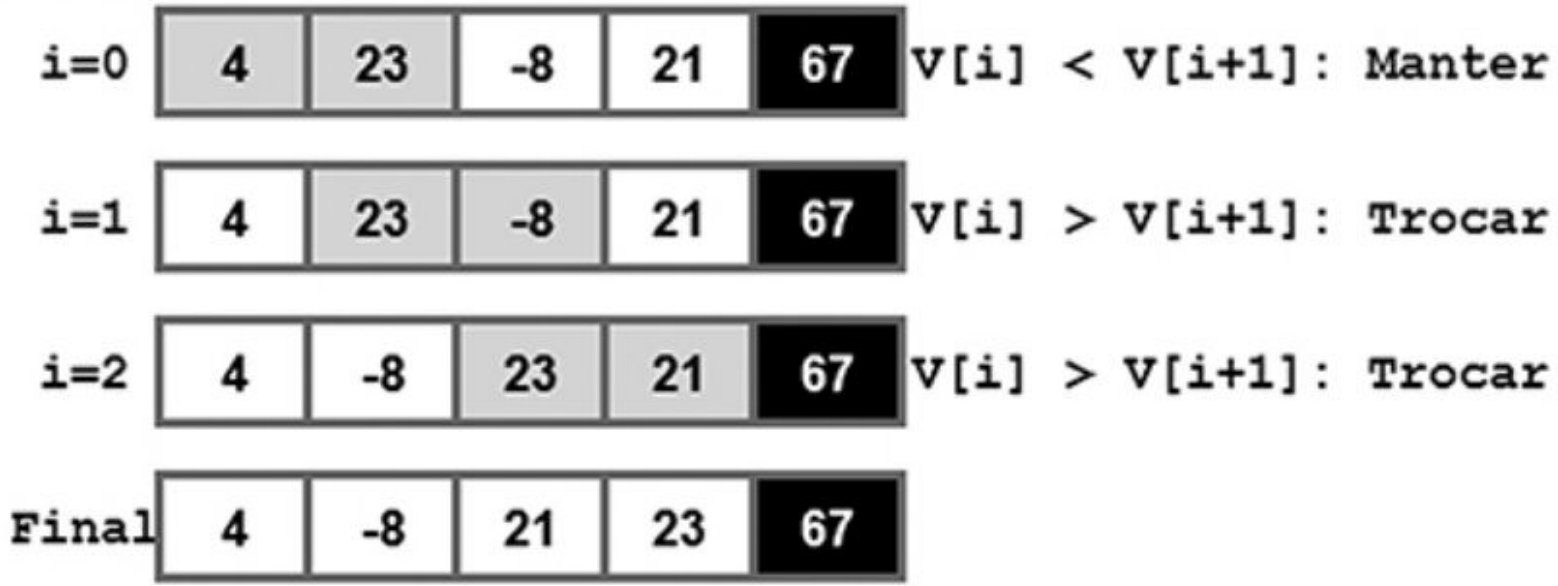
Comparar e trocar:

```
if(V[i] > V[i+1]){
   aux = V[i];
   V[i] = V[i+1];
   V[i+1] = aux;
   continua = i;
}
```

ALGORITMOS BÁSICOS DE ORDENAÇÃO - Ordenação por “bolha”: bubble sort

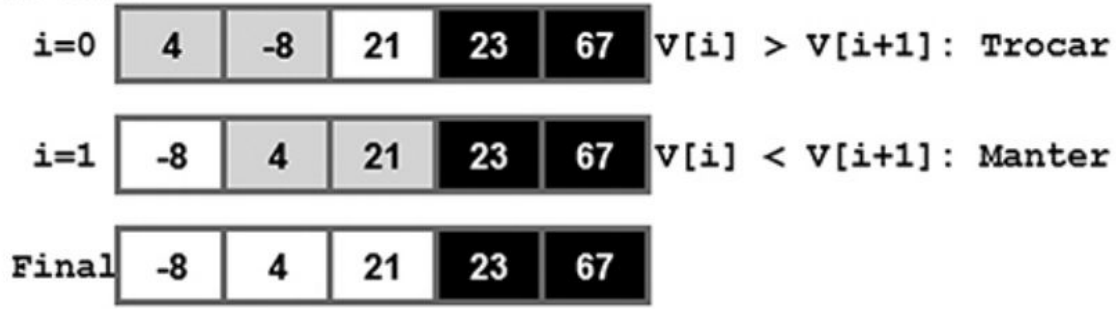
ALGORITMOS BÁSICOS DE ORDENAÇÃO - Ordenação por “bolha”: bubble sort

2º Iteração do-while



ALGORITMOS BÁSICOS DE ORDENAÇÃO - Ordenação por “bolha”: bubble sort

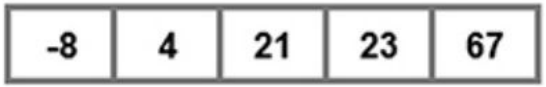
3º Iteração do-while



4º Iteração do-while

Não houve mudanças: ordenação concluída

Ordenado



ALGORITMOS BÁSICOS DE ORDENAÇÃO - Ordenação por “bolha”: bubble sort

- Sua eficiência diminui drasticamente à medida que o número de elementos no array aumenta.
- Não é recomendado para aplicações que envolvam grandes quantidades de dados ou que precisem de velocidade.
- Considerando um array com **N** elementos, o tempo de execução do bubble sort é:

$O(N)$, melhor caso: os elementos já estão ordenados.

$O(N^2)$, pior caso: os elementos estão ordenados na ordem inversa

$O(N^2)$, caso médio.

Selection Sort

- A cada passo “seleciona” o melhor elemento (maior ou menor, dependendo do tipo de ordenação) para ocupar aquela posição do array.
- O algoritmo selection sort divide o array em duas partes:
  - a parte ordenada, à esquerda do elemento analisado,
  - e a parte que ainda não foi ordenada, à direita do elemento.
  - Para cada elemento do array, começando do primeiro, o algoritmo procura na parte não ordenada (direita) o menor valor (ordenação crescente) e troca os dois valores de lugar. Em seguida, o algoritmo avança para a próxima posição do array e esse processo é feito até que todo o array esteja ordenado.

ALGORITMOS BÁSICOS DE ORDENAÇÃO - Ordenação por seleção: selection sort

Método selection sort

```
01  void selectionSort(int *V, int N){
02      int i, j, menor, troca;
03      for(i = 0; i < N-1; i++){
04          menor = i;
05          for(j = i+1; j < N; j++){
06              if(V[j] < V[menor])
07                  menor = j;
08          }
09          if(i != menor){
10              troca = V[i];
11              V[i] = V[menor];
12              V[menor] = troca;
13          }
14      }
15  }
```

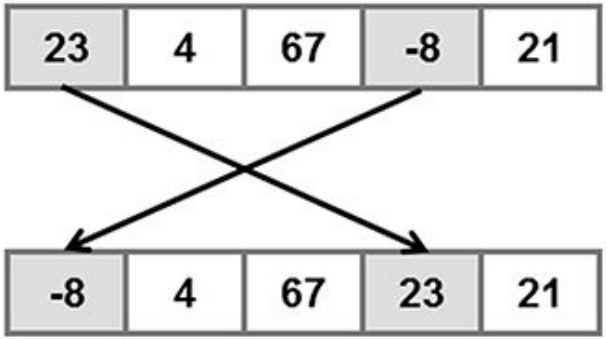
ALGORITMOS BÁSICOS DE ORDENAÇÃO - Ordenação por seleção: selection sort

Animação:

https://pt.wikipedia.org/wiki/Ficheiro:Selection_sort_animation.gif

https://www.youtube.com/watch?v=Ns4TPTC8whw

ALGORITMOS BÁSICOS DE ORDENAÇÃO - Ordenação por seleção: selection sort



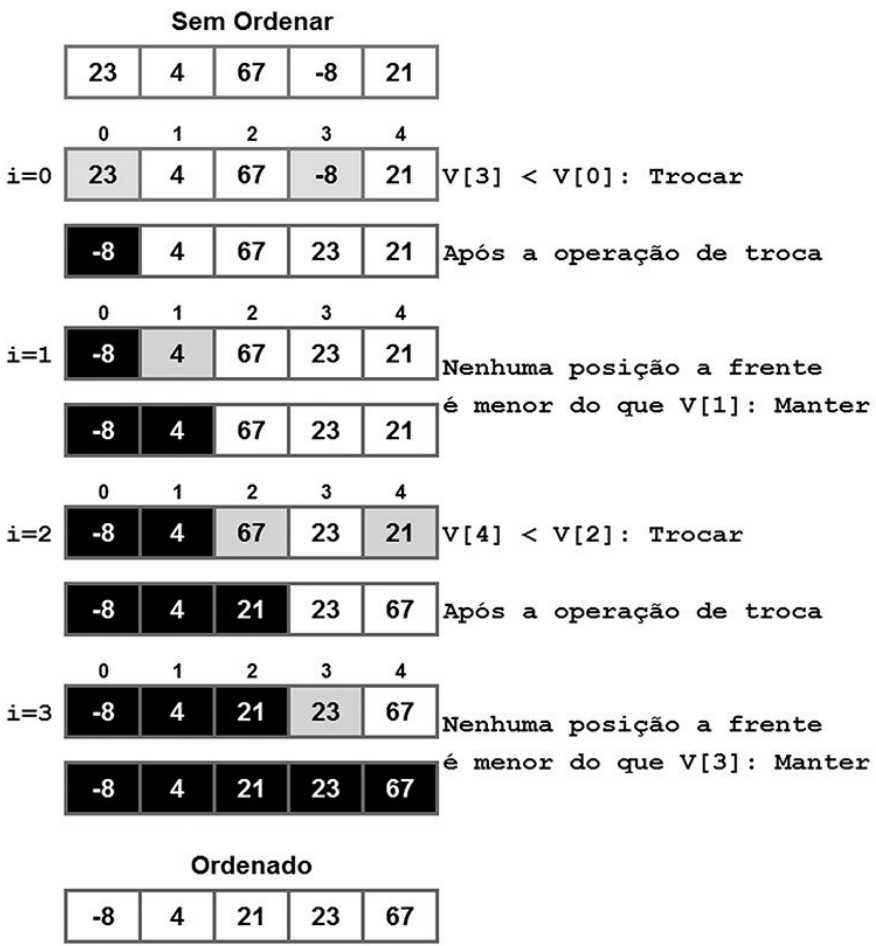
Procura o menor valor a direita:

```
menor = i;
for(j = i+1; j < N; j++){
  if(V[j] < V[menor])
    menor = j;
}
```

Troca os valores de lugar:

```
if(i != menor){
  troca = V[i];
  V[i] = V[menor];
  V[menor] = troca;
}
```

ALGORITMOS BÁSICOS DE ORDENAÇÃO - Ordenação por seleção: selection sort



ALGORITMOS BÁSICOS DE ORDENAÇÃO - Ordenação por seleção: selection sort

- Em geral, temos N iterações no laço mais externo. Cada iteração comparamos pares de elementos e trocamos tais elementos se necessário.
    - Dado um vetor de tamanho N, a primeira iteração nos dá (N-1) comparações. A segunda, (N-2), dessa forma, o total de comparações é:

$$(N-1)+(N-2)+(N-3)+.....+3+2+1=\frac{N(N-1)}{2}=\mathcal{O}(N^2)$$

- Portanto, $O(N^2)$, caso médio.
- No pior caso, o vetor está ordenado de forma inversa, onde também nos dá os dá (N-1) comparações na primeira iteração, (N-2) na segunda… Portanto, $O(N^2)$, pior caso.
- No melhor caso, onde não há nenhuma troca O(N)

ALGORITMOS BÁSICOS DE ORDENAÇÃO - Ordenação por seleção: selection sort

- O selection sort, assim como o bubble sort, não é um algoritmo eficiente.
- Eficiência diminui drasticamente à medida que o número de elementos no array aumenta
- Considerando um array com **N** elementos, o tempo de execução do selection sort é sempre de ordem $O(N^2)$.
- Como se pode notar, a eficiência do selection sort não depende da ordem inicial dos elementos.

Apesar de possuírem a mesma complexidade no caso médio, na prática, o selection sort quase sempre supera o desempenho do bubble sort, pois envolve um número menor de comparações.

Questão:

Na função selectionSort, o que acontece se trocarmos “for (i=0” da linha 3 por “for (i=1”? e se trocássemos V[j]<V[menor] da linha 6 por V[j]<=V[menor]?

Método selection sort

```
01  void selectionSort(int *V, int N){
02      int i, j, menor, troca;
03      for(i = 0; i < N-1; i++){
04          menor = i;
05          for(j = i+1; j < N; j++){
06              if(V[j] < V[menor])
07                  menor = j;
08          }
09          if(i != menor){
10              troca = V[i];
11              V[i] = V[menor];
12              V[menor] = troca;
13          }
14      }
15  }
```

Questão:

Escreva uma função que receba um inteiro `val` e um vetor `V[0..n-1]` de inteiros em ordem crescente e insira `val` no vetor de modo a manter a ordem crescente. Assuma que o vetor de entrada tem uma posição a mais.

```
int main(){
    int val=2;
    int V[5]={1, 3, 4, 5};
    insere(V, val, 5);
}
```

ALGORITMOS BÁSICOS DE ORDENAÇÃO - Ordenação por inserção: insertion sort

- Se assemelha ao processo de ordenação de um conjunto de cartas de baralhos com as mãos:
  - Pega uma carta de cada vez e a “insere” em seu devido lugar, sempre deixando as cartas da mão em ordem.
- O algoritmo insertion sort percorre um array e, para cada posição **X**, verifica se o seu valor está na posição correta.
- Isso é feito andando para o começo do array, a partir da posição **X**, e movimentando uma posição para frente os valores que são maiores do que o valor da posição **X**.
- Desse modo, teremos uma posição livre para inserir o valor da posição **X** em seu devido lugar.

ALGORITMOS BÁSICOS DE ORDENAÇÃO - Ordenação por inserção: insertion sort

Método insertion sort

```
01  void insertionSort(int *V, int N){
02      int i, j, atual;
03      for(i = 1; i < N; i++){
04          atual = V[i];
05          for(j = i; (j > 0) && (atual < V[j - 1]); j--)
06              V[j] = V[j - 1];
07          V[j] = atual;
08      }
09  }
```

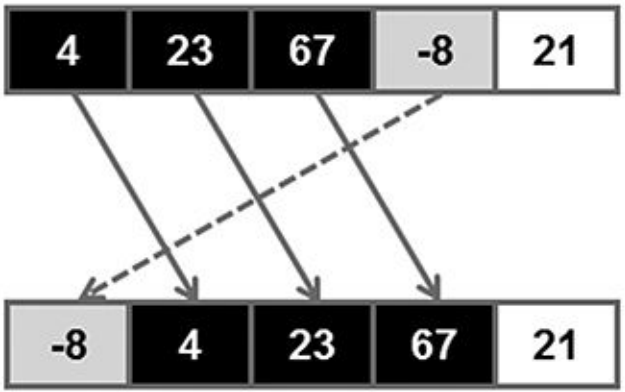
ALGORITMOS BÁSICOS DE ORDENAÇÃO - Ordenação por inserção: insertion sort

Animação:

https://pt.wikipedia.org/wiki/Ficheiro:Insertion_sort_animation.gif

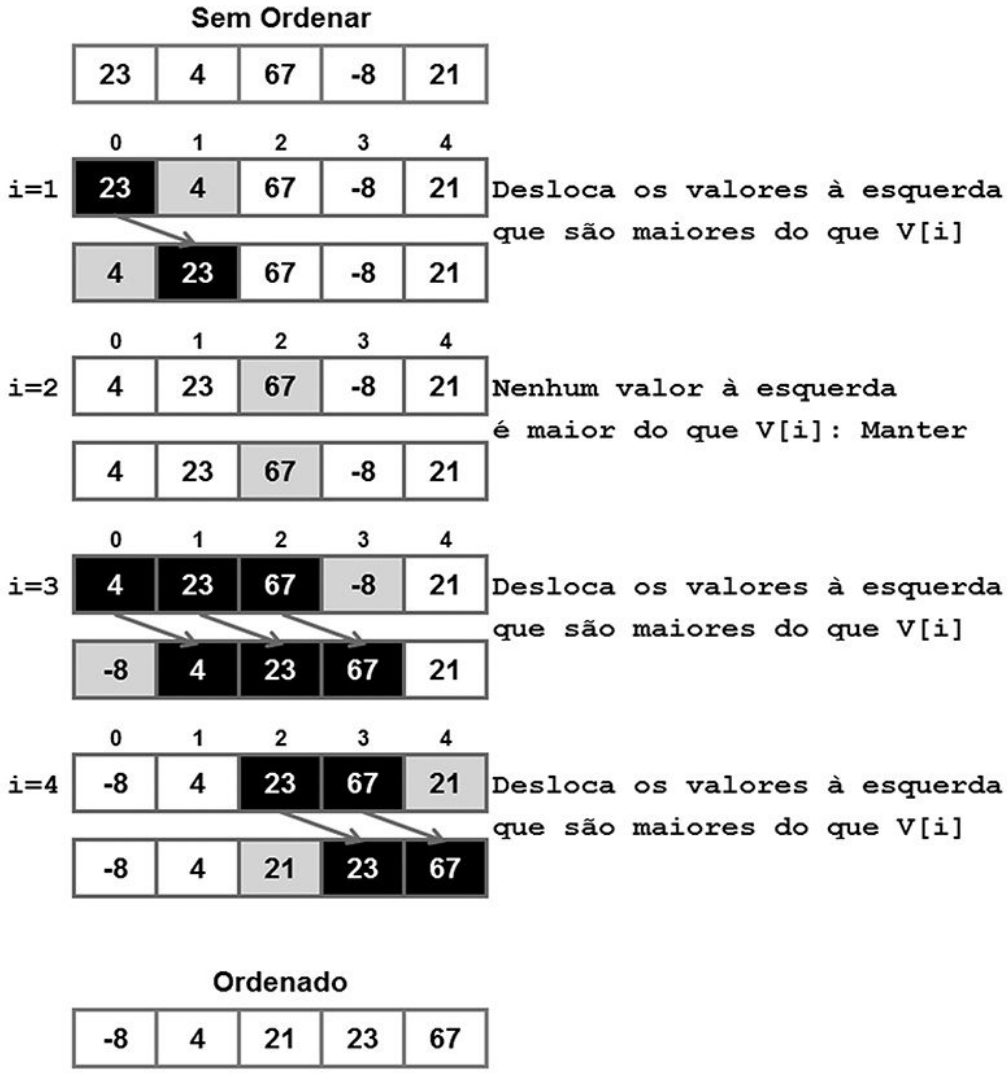
ALGORITMOS BÁSICOS DE ORDENAÇÃO - Ordenação por inserção: insertion sort

- Para i=3…
- aux=-8...



Desloca os valores a esquerda e insere:

```
aux = V[i];
for(j = i; (j > 0) && (aux < V[j - 1]); j--)
   V[j] = V[j - 1];
V[j] = aux;
```


Sem Ordenar
23 4 67 -8 21
0 1 2 3 4
i=1
23 4 67 -8 21
Desloca os valores à esquerda
que são maiores do que V[i]
4 23 67 -8 21
0 1 2 3 4
i=2
4 23 67 -8 21
Nenhum valor à esquerda
é maior do que V[i]: Manter
4 23 67 -8 21
0 1 2 3 4
i=3
4 23 67 -8 21
Desloca os valores à esquerda
que são maiores do que V[i]
-8 4 23 67 21
0 1 2 3 4
i=4
-8 4 23 67 21
Desloca os valores à esquerda
que são maiores do que V[i]
-8 4 21 23 67
Ordenado
-8 4 21 23 67

ALGORITMOS BÁSICOS DE ORDENAÇÃO - Ordenação por inserção: insertion sort

- Considerando um array com **N** elementos, o tempo de execução do insertion sort é:

  $O(N)$, melhor caso: os elementos já estão ordenados.
  $O(N^2)$, pior caso: os elementos estão ordenados na ordem inversa.
  $O(N^2)$, caso médio.

ALGORITMOS BÁSICOS DE ORDENAÇÃO - Ordenação por inserção: insertion sort

Além de ser um algoritmo de fácil implementação, o insertion sort tem a vantagem de ser:

- **Estável**: a ordem dos elementos iguais não muda durante a ordenação.
- **On-line**: pode ordenar elementos na medida em que os recebe, ou seja, não precisa ter todo o conjunto de dados para colocá-los em ordem.

Questão

- Na função insertionSort, troque a comparação `atual<V[j-1]` por `atual<=V[j-1]`. A nova função continua correta? O que acontece se trocarmos, na linha 3, “for (i=1” por “for (i=0”?

Método insertion sort

```
01  void insertionSort(int *V, int N){
02      int i, j, atual;
03      for(i = 1; i < N; i++){
04          atual = V[i];
05          for(j = i; (j > 0) && (atual < V[j - 1]); j--)
06              V[j] = V[j - 1];
07          V[j] = atual;
08      }
09  }
```

Questão

- Quantas vezes, no pior caso, o algoritmo Insertionsort copia um elemento do vetor de um lugar para outro? Quantas vezes isso ocorre no melhor caso?

Método insertion sort

```
01  void insertionSort(int *V, int N){
02      int i, j, atual;
03      for(i = 1; i < N; i++){
04          atual = V[i];
05          for(j = i; (j > 0) && (atual < V[j - 1]); j--)
06              V[j] = V[j - 1];
07          V[j] = atual;
08      }
09  }
```

Questão

- Como fazer o insertionSort ordenar de forma decrescente?

Método insertion sort

```
01  void insertionSort(int *V, int N){
02      int i, j, atual;
03      for(i = 1; i < N; i++){
04          atual = V[i];
05          for(j = i; (j > 0) && (atual < V[j - 1]); j--)
06              V[j] = V[j - 1];
07          V[j] = atual;
08      }
09  }
```

Atividade

Em grupos de até três alunos, escreva duas funções que ordenam, pela matrícula, a lista encadeada disponibilizada em http://www.facom.ufu.br/~backes/wordpress/ListaDinamicaEncadeada.zip .

1. A primeira função implementa o InsertionSort para ordenação com a função abaixo:
   int insertion_sort(Lista* li)
2. A segunda função implementa o SelectionSort para ordenação com a função abaixo:
   int selection_sort(Lista* li)

Observe que devem ser feitas manipulações dos ponteiros para os próximos elementos na própria lista de entrada. **Avalie, de forma empírica, o tempo de execução no melhor e pior caso de ambas implementações com 1000 elementos distintos.**

Referências

Estrutura de Dados descomplicada em Linguagem C (André Backes): Cap 3;

Projeto de Algoritmos (Nivio Ziviani): Capítulo 1 e 4 - 4.1.1, 4.1.2;